TECHNIQUES DEPARTMENT

NEWS

CEIR (The Council for Economic and Industry Research) has announced a program for their 704 which simulates the operation of a 650 and accepts source programs in 650 machine language. This system is described in a brochure which was distributed at the Eastern Joint Computer Conference in Washington, D. C. This should be of interest to many since it can serve to alleviate a temporary overload of 650 facilities, run programs with excessive storage requirements or act in many ways as a buffer for various emergency conditions. The full instruction repertoire is accepted and it is asserted that (on the basis of rental costs for both machines) the cost per answer when run on the simulator is at least as low as when run on a 650. Many cases have been run where the 650 program has been patched to use 704 library subroutines, greatly increasing computing speeds. At the present this program is the exclusive property of CEIR, and further information may be obtained by writing them at 1200 Jefferson Davis Highway, Arlington 2, Virginia.

The advent of this system prompted this department to make a short survey of systems which accept the machine language program of another computer and simulate the running of that program. The chart on page 4 contains all of the information it has been possible to gather from limited sources and further information is solicited from readers. From all appearances, most of these simulators will never be overpopular.

VARIABLE-WIDTH TABLES WITH BINARY-SEARCH FACILITY

MARK HALPERN, IBM Corporation

The family of subroutines described in this report was designed to create, search and maintain tables which are to contain entries of different lengths, and yet be amenable to search by partition, or "binary" search. It is designed explicitly for fixed-word-length binary machines; the tables in question are the argument-function type, with the two parts physically separated—i.e., no one machine word contains both, or parts of both. The family consists at present of seven subroutines, of which four are primitive and three second-generation. (By "primitive" is meant a self-sufficient routine; a second-generation routine is one which calls on one or more primitives.) These routines are at present coded for the IBM 709; with a few trivial changes they are ready also for the 704. The primitives are: (1) STT (Start Table), (2) TLU (Table Look-Up), (3) ITC (Increase Table Capacity), and (4) AOD (Append Ordered Data). The second-generation routines are: (1) INT (Insert in Table), (2) INX (Insert and Expand), and (3) ANX (Append and Expand). It is intended that this family be written, with modifications as necessary, for several stored-program computers. The calling-sequences for any one of the routines will, as far as possible, be identical on the several machines.

The tables dealt with by this family are organized in the form of matrices, or series of sub-tables, as shown in Figure 1.

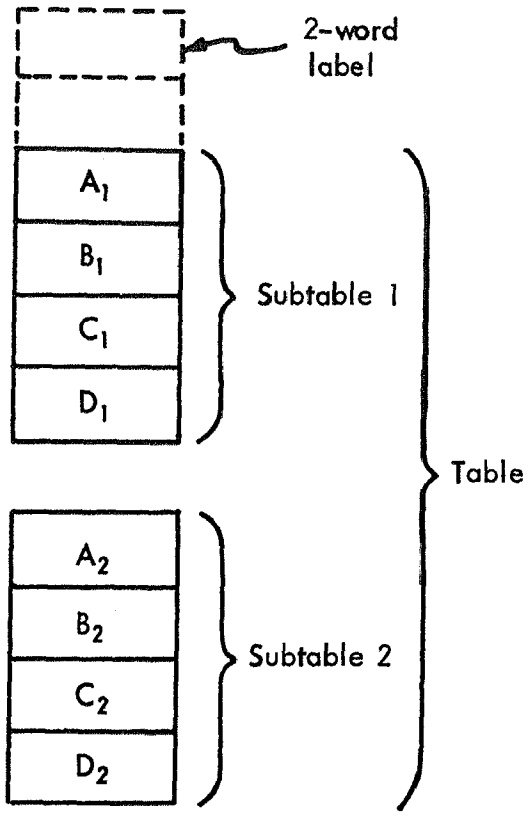

FIGURE 1

Figure 1 shows a table in which 4 entries have been made: A, B, C and D. Each is too long to fit into a single word, and has accordingly been divided in two. The first parts, A_1, B_1, C_1 and D_1, have been collected and stored in subtable 1; the second parts, A_2, B_2, C_2 and D_2, in subtable 2. The order of entries in subtable 1 is determined by the values of the whole entries A, B, C and D; the order in subtable 2 is determined solely by the locations in subtable 1 of the high-order parts of these entry-segments. Only subtable 1 entries, then, will necessarily exhibit monotonic increase or decrease in value. There may be as many such subtables as are required, and all succeeding subtables will be ordered as was subtable 2. If A, B, C and D are not of equal length, these rules hold: (a) the number of subtables required is that required by the longest entry, and (b) trailing zero's will be attached to shorter entries to bring them up to the length of the longest.

When this table is to be searched, the first word's worth of the comparand (the value for which the table is being searched) is isolated, and a binary search of subtable 1 is made until either a match is found, or one of several special conditions, to be discussed below, is detected. Assume a match found at the Nth location of subtable 1, which we will call the base-point. The second word's worth of the comparand is then matched against the Nth word of subtable 2. If there are more than two subtables, the process is continued, successive words' worth of the comparand being matched against the Nth locations of successive subtables. If matches are obtained straight through to the end, the search has been successfully completed. If at any stage after the fixing of a base-point a match is not obtained, a new base-point is defined, equal to the old base-point plus or minus one. (The new base-point will be higher or lower as the no-match condition dictates.) If the value at the new base-point is equal to that of the first word's worth of the comparand, the process of attempting matches between successive words' worth of the comparand and Nth entries (N now being the ordinal number of the new base-point) of successive subtables may proceed. This process may be repeated as often as no-match conditions require, provided only that the new base-point required by the no-match condition is equal in value to the first word's worth of the comparand.

This process must eventually end in one of two ways: the finding of an entry which matches the comparand word-for-word, or the finding of a pair of adjacent entries which bound the value of the comparand. This pair of values may be used for interpolation if the function is continuous, or, if the function is discrete, simply as an indication that the comparand has no equivalent in the table. The possibility that the value of the comparand lies outside the upper or lower bound of the table is eliminated at the outset by attempting a preliminary match between the comparand and the least and greatest argument in the table. This technique has proven to be highly economical of time. Comparands falling outside the table can be detected and reported on before even the initialization for the subroutine has been completed.

Assuming the comparand matched perfectly with some argument in the table, the function may then be obtained in either of two ways. If the number of sub-tables required by the arguments is Y, then a (Y+1)th sub-table will contain either the first words of the functions associated with the arguments stored above, or simply a series of addresses giving the locations of the functions. If the first method is adopted, then functions longer than one word may be broken up and stored in successive sub-tables just

as the arguments were. Table-searching will involve only the first Y sub-tables, of course; table modification will involve Y+Z, where Z is the number of sub-tables required to store the functions or their addresses.

The sub-tables, as indicated in Figure 1, are not necessarily contiguous. Let the number of words per sub-table be N, and the number of words between Nth entries of adjacent sub-tables be Δ; then $\Delta \geq N$. (See Figure 2). Delta is a programmer-supplied parameter, and may be used to exert various kinds of control over table operations. If, for example, the programmer wishes to ensure that the table not exceed a certain size, Q, he can do so by setting

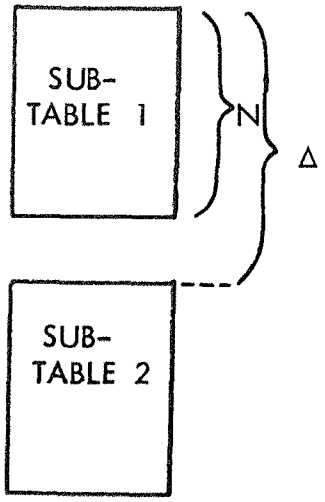

FIGURE 2

$$\Delta = \frac{Q - 2}{Y + Z},$$

and using subroutine INT to add entries to the table. This subroutine compares N against Δ before adding new entries, and transfers to an error-return if an attempt is made to add entries when $N = \Delta$. This technique also optimizes on time, since a pre-set constant Δ, combined with subroutine INT, means that the sub-tables are not to be moved further apart.

Where the programmer does not know how many entries to expect, or where space rather than time is to be optimized on, Δ should be set to a value no greater than the minimum number of entries expected (if no estimate of this number can be made, set $\Delta = 1$), and subroutine INX (Insert and Expand) must be used. This subroutine, on finding that $N = \Delta$, increases Δ by spacing the sub-tables out to intervals equal to the number of entries to be added. The parameters for INX include an absolute limit to expansion, enabling the programmer to ensure that the table cannot grow to the point where it begins to wipe out wanted information.

Where the entries to be added consist of arguments which are (a) all greater in value than the highest currently stored in the table, and (b) ordered among themselves, the subroutines to use are AOD (Append Ordered Data) instead of INT, and ANX (Append and Expand) instead of INX. These Append subroutines, as their names imply, simply attach the new entries to the bottom of the sub-tables, skipping the space-and-time-consuming look-ups which are necessary for inserting entries of unknown value.

The family of subroutines which has been described is not complete: new members being worked on will provide for automatic interpolation, both linear and parabolic and for the deletion of unwanted entries from the table.

COMPUTER BEING SIMULATED		COMPUTER ON WHICH SIMULATOR IS RUN		Real-time[1] Speed ratio —TO—		LIMITATIONS	PRODUCED BY	Date Complete
650	All models	704	8096 word memory, 6 tape units, off-line equipment	5–8	1	None, operationally (Program not available for release now)	Council for Industry and Economic Research, 1200 Jefferson Davis Highway, Arlington 2, Virginia	Fall 1957
709	2500 or 6500 words n tapes	704	4K or 8K memory n tapes	1	30	Will not simulate timed I-O DSC transfers, BTT, etc.	IBM Applied Programming at Mass. Inst. of Technology	Spring 1958
704		701	4096 word memory	1	15–120[2]	3000 word program, no Copy instr., limited I-O	United Aircraft Corp., Hartford	Spring 1955
704		701	2048 word memory Drum	1	100–500[2]	Some	RAND Corp., contributed at third SHARE meeting	Fall 1955
650	2000 word drum read-punch unit	701		1			Chance-Vought Aircraft	
650	2000 word drum read-punch unit	705	20,000 char. mem. on-line rdr, punch	1	2–4		Standard Oil of Indiana. Available as IBM Form 32-7763	Spring 1957
705	TCU	705	TRC 20K or 40K memory	1	3–8	—	IBM Applied Programming (to be replaced by auto-conversion prog.)	Fall 1957
702	10K or 20K mem.	705	20K or 40K memory	1	4–16	Some	IBM Applied Programming (See 705 Bulletin No. 6)	Spring 1957
705	20K or 40K mem.	702	10K memory Drum	1	6–20		IBM Applied Programming (See 705 Bulletin No. 6)	Summer 1955
650	Core mem., drum, indexing, flt. point	650	2000 word drum read-punch unit	1	10–20		IBM—650 Library No. 2.0.011 (Fiasco)	
650		Datatron 205		1				
Univac II	2000 word memory	Univac I	1000 word memory	1	100–150	Limited to 700 word program at any one time.	Remington-Rand. 2 versions, by Philadelphia and New York offices.	Winter 1956
1103		Univac I		1				

COMPUTER SIMULATION SYSTEMS—MACHINE LANGUAGE

[1]With respect to running of computer being simulated

[2]When program is mostly floating point arithmetic